客户化JSP标签

- 教学目标
  - 理解客户化JSP标签的作用
  - 了解JSP Tag API
  - 掌握创建并运用客户化JSP标签的步骤
  - 掌握在客户化JSP标签中访问application、session、request和page范围内的共享数据的方法。

客户化JSP标签的作用

- 客户化JSP标签技术是在JSP 1.1版本中才出现的，它支持用户在JSP文件中自定义标签，这样可以使JSP代码更加简洁。
- 这些可重用的标签能处理复杂的逻辑运算和事务，或者定义JSP网页的输出内容和格式。
- 参考itemdetail.jsp

创建客户化JSP标签的步骤

- （1）创建标签的处理类
- （2）创建标签库描述文件
- （3）在web.xml文件中声明引用的标签
- （4）在JSP文件中引入标签库，然后插入标签，例如:<mm:hello/>

JSP Tag API

- Servlet容器编译JSP网页时，如果遇到自定义标签，就会调用这个标签的处理类。
- 标签处理类必须扩展以下两个类之一：
  - javax.servlet.jsp.TagSupport
  - javax.servlet.jsp.BodyTagSupport

TagSupport类的主要方法

- doStartTag

  Servlet容器遇到自定义标签的起始标志时调用该方法

- doEndTag

  Servlet容器遇到自定义标签的结束标志时调用该方法

TagSupport类的主要方法

- setValue(String k,Object o)

  在标签处理类中设置key/value

- getValue(String k)

  在标签处理类中根据参数key返回匹配的value

- removeValue(String k)

  在标签处理类中删除key/value

TagSupport类的主要方法

- setPageContext(PageContext pc)

  设置PageContext对象，该方法由Servlet容器在调用doStartTag或doEndTag方法前调用

- setParent(Tag t)

  设置嵌套了当前标签的上层标签的处理类，该方法由Servlet容器在调用doStartTag或doEndTag方法前调用

- getParent()

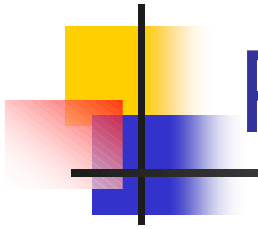
  返回嵌套了当前标签的上层标签的处理类

TagSupport类的两个重要属性

- parent：代表嵌套了当前标签的上层标签的处理类
- pageContext：代表Web应用中的javax.servlet.jsp.PageContext对象

TagSupport类的两个重要属性

- JSP容器在调用doStartTag或doEndTag方法前，会先调用setPageContext和setParent方法，设置pageContext和parent。
- 在doStartTag或doEndTag方法中可以通过getParent方法获取上层标签的处理类；在TagSupport类中定义了protected类型的pageContext成员变量，因此在标签处理类中可以直接访问pageContext变量。



PageContext类

- PageContext类提供了保存和访问Web应用的共享数据的方法：
  - public void setAttribute(String name, Object value, int scope)
  - public Object getAttribute(String name, int scope)

PageContext类（续）

- 其中，scope参数用来指定属性存在的范围，它的可选值包括：
  - PageContext.PAGE_SCOPE
  - PageContext.REQUEST_SCOPE
  - PageContext.SESSION_SCOPE
  - PageContext.APPLICATION_SCOPE
- 例如：

```
pageContext.setAttribute(“username”,”weiqin”,
   PageContext.SESSION_SCOPE);
```

TagSupport类的处理标签方法

- public int doStartTag() throws JspException
- public int doEndTag() throws JspException

doStartTag()方法

- 当Servlet容器遇到自定义标签的起始标志，就会调用doStartTag()方法。
- doStartTag()方法返回一个整数值，用来决定程序的后续流程。它有两个可选值：
  - Tag.SKIP_BODY
  - Tag.EVAL_BODY_INCLUDE

doStartTag()方法

- Tag.SKIP_BODY表示标签之间的内容被忽略。
- Tag.EVAL_BODY_INCLUDE表示标签之间的内容被正常执行。例如对于以下代码：

```
<prefix: Mytag>
   Hello
   ......
   ......
</prefix:Mytag>
```

假若<Mytag>的doStartTag()方法返回Tag.SKIP_BODY，"Hello"字符串不会显示在网页上；若返回Tag.EVAL_BODY_INCLUDE，"Hello"字符串将显示在网页上。

doEndTag()方法

- 当Servlet容器遇到自定义标签的结束标志，就会调用doEndTag()方法。
- doEndTag()方法也返回一个整数值，用来决定程序后续流程。它有两个可选值：
  - Tag.SKIP_PAGE
  - Tag.EVAL_PAGE

doEndTag()方法

- Tag.SKIP_PAGE表示立刻停止执行JSP网页，网页上未处理的静态内容和JSP程序均被忽略，任何已有的输出内容立刻返回到客户的浏览器上。
- Tag.EVAL_PAGE表示按正常的流程继续执行JSP网页。

用户自定义的标签属性

- 在标签中还能包含自定义的属性，例如：

```
<prefix:mytag username="weiqin">
 ......
 ......
</prefix:mytag>
```

用户自定义的标签属性

- 在标签处理类中应该将这个属性作为成员变量，并且分别提供设置和读取属性的方法，假定以上username为String类型，可以定义如下方法：

```
private String username;
public void setUsername(String value){
    this.username=value;
}
public String getUsername(){
    return username;
}
```

范例1：创建hello标签

- 定义一个名为mytaglib的标签库，它包含一个简单的hello标签，这个标签能够将JSP页面中所有的<mm:hello/>解析为字符串“hello”。

hello标签的处理类HelloTag

```
public int doEndTag() throws JspException {
    try {
        // We use the pageContext to get a Writer
        // We then print the text string Hello
        pageContext.getOut().print("Hello");
    }
    catch (Exception e) {
        throw new JspTagException(e.getMessage());
    }
    return EVAL_PAGE;
}
```

创建hello标签的标签库的描述文件

- 创建Tag Library的描述文件mytaglib.tld文件，在这个文件中定义mytaglib标签库和hello标签。这个文件存放位置为/WEB-INF/mytaglib.tld。

在web.xml文件中加入<taglib>元素

```
<web>
    ......
    <taglib>
      <taglib-uri>/mytaglib</taglib-uri>
      <taglib-location>/WEB-INF/mytaglib.tld</taglib-location>
    </taglib>

<web>
```

在JSP文件中加入hello标签

- （1）在 hellowithtag1.jsp 中加入引用mytaglib的taglib指令：

```
<%@ taglib uri="/mytaglib" prefix="mm" %>
```

以上taglib指令中，prefix属性用来指定引用mytaglib标签库时的前缀。

在JSP文件中加入hello标签

- （2）在hellowithtag1.jsp文件中插入hello标签：

```
<b><mm:hello/>:
<%= request.getParameter("USER") %></b>
```

- 访问hellowithtag1.jsp：

http://localhost:8080/helloapp/hellowithtag1.jsp?USER=weiqin

范例2：创建message标签

- 创建一个能替换helloapp应用中JSP网页的静态文本的标签，这个标签名为message，它放在mytaglib标签库中。

范例2：创建message标签

- 在hellowithtag2.jsp文件中使用message标签的代码如下：

```
<b><mm:message key="hello.hello" />:
<%= request.getAttribute("USER") %></b>
```

- 当客户访问hello.jsp网页时，message标签的处理类会根据属性key的值从一个文本文件中找到与key匹配的字符串。假定这个字符串为“Hello”，然后将这个字符串输出到网页上。

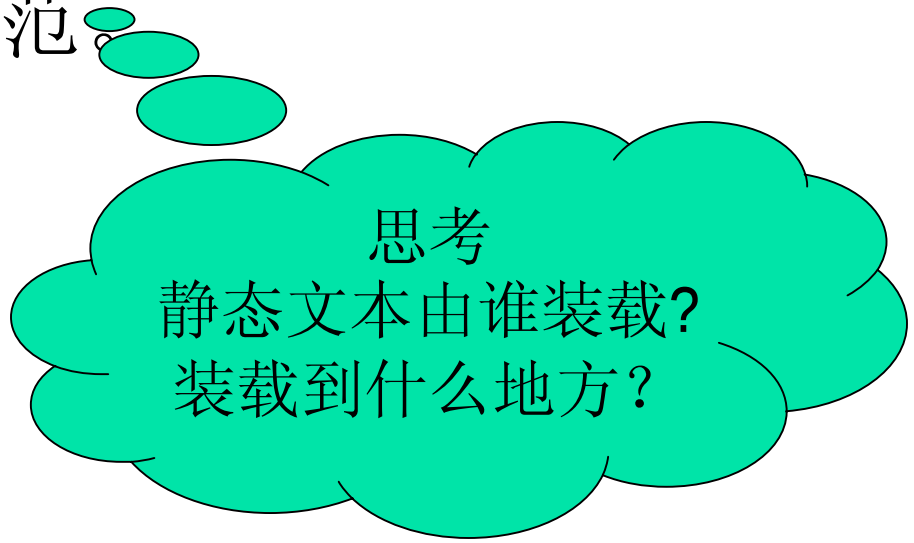
创建包含JSP网页静态文本的文件

- 首先将创建包含JSP网页静态文本的文件，这些文本以key/value的形式存放，这个文件名为messageresource.properties:

```
hello.title = Title of hello.jsp
hello.hello = Hello
```

在Web应用启动时装载静态文本

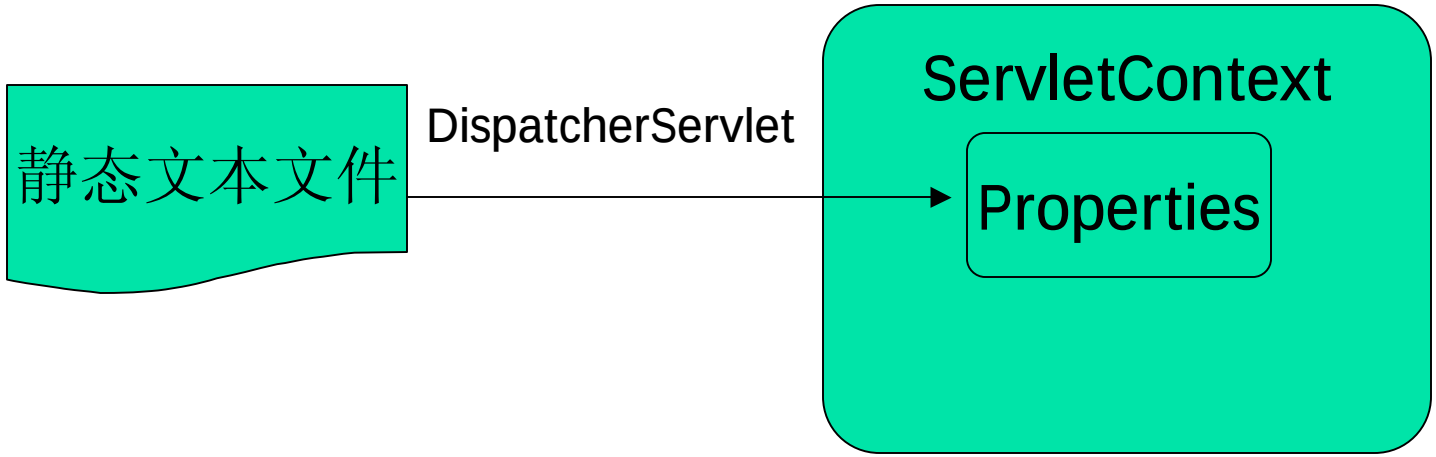
- 尽管装载静态文本的任务可以直接由标签处理类来完成，但是把初始化的操作安排在Web应用启动时完成，这更符合Web编程的规范。

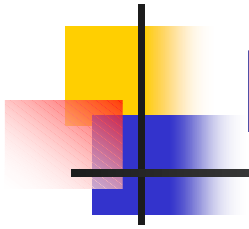


在Web应用启动时装载静态文本

- 在本例中，由DispatcherServlet类的init方法负责从静态文本文件中读取静态文本，然后把它们装载到Properties对象中，最后再把这个Properties对象作为属性保存到ServletContext中。

在Web应用启动时装载静态文本





DispatcherServlet类的init方法

```
public void init(ServletConfig config)
    throws ServletException {
    super.init(config);

    Properties ps=new Properties();
    ...
```

DispatcherServlet类的init方法

```
ServletContext context=config.getServletContext();
InputStream in=context.getResourceAsStream(
                        "/WEB-INF/messageresource.properties");
ps.load(in);
in.close();
context.setAttribute("ps",ps);
```

在Web应用启动时装载静态文本

- 为了保证在Web应用启动时就加载DispatcherServlet，应该在web.xml中配置这个Servlet时设置load-on-startup属性：

```
<servlet>
    <servlet-name>dispatcher</servlet-name>
    <servlet-class>mypack.DispatcherServlet</servlet-class>
    <load-on-startup>1</load-on-startup>
</servlet>
```

创建MessageTag标签处理类

- MessageTag包含一个成员变量key，它与message标签的属性key对应。在MessageTag中定义了getKey和setKey方法：

```
private String key=null;
    public String getKey(){
      return this.key;
}
public void setKey(String key){
      this.key=key;
}
```

创建MessageTag标签处理类

- 在MessageTag的doEndTag方法中，首先从pageContext中读取包含静态文本的Properties对象：

```
Properties ps=
    (Properties)pageContext.getAttribute("ps",
    pageContext.APPLICATION_SCOPE);
```

创建MessageTag标签处理类

- 然后从Properties对象中读取key对应的静态文本，最后输出该文本：

```
String message=null;
message=(String)ps.get(key);
pageContext.getOut().print(message);
```

在mytaglib库中定义message标签

```
<tag>
    <name>message</name>
    <tagclass>mypack.MessageTag</tagclass>
    <bodycontent>empty</bodycontent>
    <info>produce message by key</info>
    <attribute>
     <name>key</name>
     <required>true</required>
    </attribute>
</tag>
```

练习题1

- 问题：在标签处理类中，如何访问session范围内的共享数据?
- 选项：
  - (A)在TagSupport类中定义了session成员变量，直接调用它的getAttribute()方法即可。
  - (B)在标签处理类TagSupport类中定义了pageContext成员变量，先通过它的getSession()方法获得当前的HttpSession对象，再调用HttpSession对象的getAttribute()方法。
  - (C)pageContext.getAttribute("attributename",PageContext.SESSION_SCOPE)
- 答案: B,C

练习题2

- 问题：在下面的选项中，哪些是TagSupport类的doStartTag()方法的有效返回值?
- 选项：
  - (A) Tag.SKIP_BODY
  - (B) Tag.SKIY_PAGE
  - (C) Tag.EVAL_BODY_INCLUDE
  - (D) Tag.EVAL_PAGE
- 答案: A,C